An Integrated Modeling and Simulation Methodology for Intelligent Systems Design and Testing

Xiaolin Hu and Bernard P. Zeigler
Arizona Center for Integrative Modeling and Simulation
The University of Arizona
Tucson, AZ, USA 85721
www.acims.arizona.edu

ABSTRACT

Model continuity refers to the ability to use the same model of a system throughout its design phases. For intelligent systems, we can restrict such continuity to the intelligent control components, and more specifically, the models that implement the system's decision making. behavior. In this paper, we show how a modeling and simulation environment, based on the DEVS formalism,, can support model continuity in the design of intelligent systems. For robotic systems, such continuity allows design and testing of the same control logic model through the phases including logical simulation, real-time simulation and actual execution.

KEYWORDS: *Model Continuity, Modeling, Simulation, Experimental Frame, Real Time Systems, Intelligent Systems, DEVS*

1. INTRODUCTION

One criterion for intelligence is the ability to make decisions in a timely manner. Certainly for systems expected to interact with the real world, such as robotic systems, real time constraints play a major role, although they may vary in stringency for different behaviors. With the rapid advance in processor speed, memory capacity, sensors and actuators, and dramatic increases in network technology, intelligence has a natural association with distributed systems, as exemplified by multi-agent systems. Unfortunately, the lack of good design methods and support tools has made software development for intelligent systems a bottleneck. To address the importance and complexity of real time software development, academic and commercial tool developers have proposed various real time software models and methods that represent different emphases on this problem. However, so far none of them fits very well to support real time software from a systematic way. A formal methodology is needed for real-time software development [1, 2]. The method should support software development for intelligent systems including designing, testing and execution in a systematic way, with a framework to integrate a system's behavior, structure and timeliness together.

In this paper, we describe an approach to develop real time software for intelligent systems. This approach is based on DEVS modeling and simulation framework [3]. Corresponding to the general "Design—Test—Execute" development procedure, our approach provides a "Modeling—Simulation—Execution" methodology which includes several stages to develop real time software. In the modeling stage, Atomic and Coupled models are built to capture a system's behavioral and structural properties. In the simulation stage, a series of simulators is chosen to simulate and test model's behavior in an incremental fashion step. In the execution stage, the verified model is executed by real-time execution engine. It is important to point out that during the whole process, we maintain model continuity because the same model that has been designed will be simulated and then executed. For distributed systems, this continuity also means the coupling among the models is maintained even though the models are executed in a distributed environment. We believe keeping model's continuity is an efficient way to manage software's complexity and consistency. With model's continuity, we are confident that the system in operation is the system we wanted to design and will carry out the functions as tested by simulation.

This paper will start with the description of the methodology for a stand-alone real time system. Then it will scale up to distributed real time systems. For both systems, step-wise simulation methods are provided to simulate and exercise the model under test. Finally, we describe how experimental frame, a more general testing environment, can be integrated to test the model of interest while still preserving model continuity.

2. MODELING, SIMULATION AND MODEL CONTINUITY

Intelligent real time systems monitor, respond to, or control, an external environment. This environment is connected to the digital logic through sensors, actuators, and other input-output interfaces [4]. A real time system from this point of view consists of sensors, actuators and the real time control and information-processing unit. For simplicity, we will call this last one the control model. The sensors get inputs from the real environment and feed them to the control model. The actuators get commands from the control model and perform corresponding actions to affect the real environment. The control unit processes the input from sensors and makes decisions based on its control logic. Depending on the complexity of the system, the control model could have only one central component or it could have multiple parallel-processing subcomponents, which in turn may have their own sub-control units.

Once we establish this view of a real time system as shown on the left side of Figure 1, we can model it easily. In our approach, sensors and actuators are modeled as DEVS Activities, which is a concept introduced by RT-DEVS for real time system specification [5]. A DEVS Activity can be any kind of computer task. However, in the context of this paper, we only consider the sensor/actuator Activities. The control unit is modeled as a control model which might has a set of subcomponents. These subcomponents are coupled together so they can communicate and cooperate. With this approach, the control model acts as the brain to process data and make decisions. It could be a simple Atomic model or a complex hierarchical coupled model. Sensor/actuator Activities act as hardware interfaces providing a set of APIs for the control model to use. They are essentially hardware drivers for sensors and actuators. How to define an Activity and its APIs is dependent on how the designer delineates the "Control Model—Activity" boundary. For example, we can model a sensor module that may have its own control logic as a sensor Activity. Or we can also include that part of logic into our control model and only model the sensor hardware as an Activity. The clear separation between control model and activity's functions makes it possible for the designer to focus on his design interest. In the context of intelligent real time systems, the control logic is typically very complex, as the system usually operates in a dynamic, uncertain or even hostile environment. As such, the control model is the main interest of design and testing. In our approach, simulation methods are applied to test the correctness and efficiency of this model. Tne continuity of this model is also emphasized during the whole process of the methodology.

Before simulating the control model, we need to model the real environment as an environment model. This environment model is a reflection of how the real environment affects or is affected by the system under design. Meanwhile, a "simulated" sensor/actuator hardware interface is also needed for the control model to talk to the environment model. This is why we introduce the SimActivity concept. In contrast to an Activity, which drives real hardware and is actually being executed, a SimActivity imitates an Activity's interface/behavior and

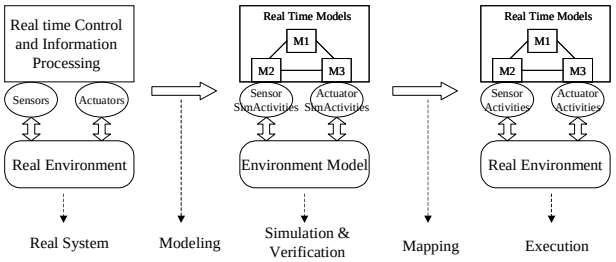

Figure 1: Modeling, Simulation and Execution of Non-distributed Real Time System

is only used during simulation. A sensor SimActivity gets input from the environment model just as a sensor Activity gets input from the real environment. An actuator SimActivity does similar things as an actuator Activity too. Note that it is important for an Activity and its SimActivity to have the same interfaces, which are used by the control model in both simulation and real execution. By imposing this restriction, the control model can be kept unchanged in the transition from simulation to execution (it interacts with the environment model and real environment using the same interfaces). Thus, model continuity is achieved.

As shown in the center of Figure 1, in the modeling stage, a simulated system is developed based on the real system. With this system, different simulation strategies can be applied to validate the control model. In DEVS, there is a clear separation between a model and its simulators, which gives us the flexibility to choose different simulators to simulate the same model. These simulators include fast-mode simulator, real-time simulator and distributed simulators. With these simulators, a model can be simulated and tested incrementally before its real execution. During the simulation stage, employing fast-mode (or logical time) simulators, if we find the simulated result is not what we

expected, the model can be revised and then re-simulated. This "modeling-simulation-revising" cycle repeates until we are satisfied with simulation result or nothing more can be learned in the simulation stage. A more detailed description of how to choose and use different simulators is given in the next section.

After the model is validated through simulations, it will be mapped to the real hardware for execution. For a non-distributed application, this mapping is the "Activity Mapping" to associate the sensor/actuator Activities to the corresponding sensor/actuators hardware. For a distributed application, an extra "Model Mapping" is needed to map a set of cooperative models to a set of networked nodes. By associating the models and Activities to their corresponding hardware, the system can be executed in a real environment. In execution, the control logic is governed by the control model, which has been validated in the step-wise simulation. If true model continuity from simulation to execution has been achieved, this control model will carry out the control logic just the same as it did when simulated. In practice, one may not be able to completely replicate the real environment in the environment model, and there will be potential for design problems to surface in real execution. When this happens, re-iteration through the stages can be more easily achieved with the model continuity approach.

3. STEP-WISE SIMULATION AND TESTING

Simulation technology has been widely applied to help to design and test real time systems. This technology provides a valuable tool for engineers to test and understand the system under design. When the complexity of a problem is too large to allow an analytical solution, simulation is the only option to investigate system configurations or operational modes prior to the implementation in the field. In this section, we will show how different simulation methods can be applied to incrementally simulate and test a stand-alone system. Simulations for distributed systems will be shown in section 5.

As shown in step 1 of Figure 2, for a stand-alone system, three different simulation steps can be applied to test the model. They are fast-mode simulation, real-time simulation and hardware-in-the-loop simulation. These simulation methods apply different simulation configurations to test different aspects of the model under test.

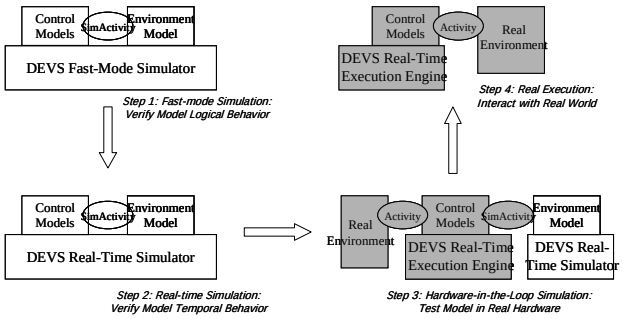

Figure 2: Step-wise Simulations of Non-distributed Real Time System

In fast-mode simulation, the control model is configured to talk to the environment model through sensor/actuator SimActivities. These models stay in one computer and a DEVS fast-mode simulator is chosen to simulate them. In fast-mode simulation, the flow of time is logical (not connected to a wall-clock). So, a fast-mode simulator generates simulation results as fast as it can. Based on these results, we can analyze the data to see if the system under test fulfills the logical behavior as desired.

Just as fast-mode simulation verifies a model's logical behavior, real-time simulation verifies model's temporal behavior. In real time simulation, the model's setting is the same as in fast-mode simulation. However, the fast-mode simulator is replaced by a real-time simulator, which executes the model at the same speed as a real world clock. Since the simulation runs in real time, we can test a model's temporal behavior such as checking if critical deadlines can be met.

In fast-mode and real-time simulations, the model under test and the simulators reside in one computer. This computer is not the same computer as the one in which the model will actually be executed. Instead, a simulated environment is provided. However, not all components in a complex system can be modeled in adequate detail in computer simulation. Sometimes, the executing hardware can have significant impact on haw well a model's functions can be carried out. For example, processor speed and memory capacity are two typical factors that can affect the performance of an execution. Thus, to make sure that the control model, having been validated in fast-mode and in real-time simulation, also can execute correctly in the real hardware, we adopt the hardware-in-the-loop (HIL) simulation [6,7]. As shown in step 3 of Figure 2, in HIL simulation, the environment model is simulated by a DEVS real time simulator on one

computer. The control model under test is executed by DEVS real-time execution engine on the real hardware. This DEVS real-time execution engine is a stripped-down version of DEVS real-time simulator. It provides a compact and high-performance runtime environment to execute DEVS models. In HIL simulation, the model under test interacts with the environment model through SimActivities. These SimActivities act as simulated sensors or actuators. Real sensors or actuators can also be included into HIL simulation by using sensor/actuator Activities. The decision of which sensor/actuator will be real hardware and which sensor/actuator will be simulated SimActivities is dependent on the test engineer's testing objectives. With different testing objectives, different combinations of real sensor/actuators and simulated sensor/actuators can be chosen to conduct an exhaustive test of the control model. Notice that in HIL simulation, as the control model and environment model stay on different computers, a bi-directional connection must be established between the two computers. We use LAN connection based on TCP/IP protocol because it is widely used in industry, can sustain high-speed data transfer, and very portable. This connection is taken care of by DEVS real-time simulator and execution engine so it is transparent to the model.

Once we passed hardware-in-the-loop simulation, we are ready to leave the simulation stage for execution stage. As shown in step 4 of Figure 2, in real execution, DEVS real-time execution engine executes the control model. There is no environment model because the control model will interact with the real environment through the sensor/actuator Activities.

4. DISTRIBUTED REAL TIME SYSTEMS

With the advance of network technology, distributed real time systems are playing more and more important roles. Figure 3 shows an example distributed system with three nodes. Generally speaking, a distributed real time system consists of a set of subsystems. Like a stand-along system, each subsystem has its own control and information processing unit and it interacts with the real environment through Sensor/Actuators. However, these subsystems are not "along". They are physically connected by network and logically they talk to each other and cooperate to finish a common task. Distributed real time systems are much harder to designed and tested because one subsystem's behaviors may affect one or all of other subsystems. These subsystems influence each other not only by explicit communications, but also by implicit environment change as they all share the same environment. For example, in Figure 3, if Node 1 changes the environment through its actuators, this change will be seen by the sensors of Node 2, thus affects Node 2's decision making. With this kind of influence property, it's not practical to design and test each subsystem separately and then put them together. Instead, the system as a whole needs to be designed and tested.

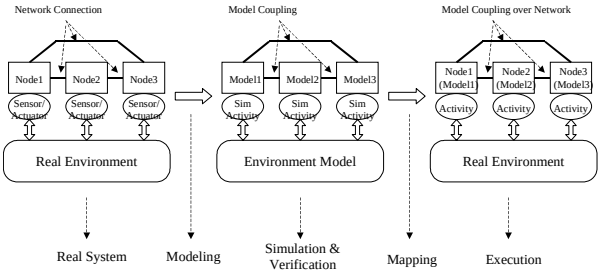

Figure 3: Modeling, Simulation and Execution of Distributed Real Time System

In our approach, a distributed real time system is modeled as a coupled model. This coupled model consists several subcomponents. Each subcomponent is corresponding to a subsystem of the distributed real time system. As described in section 2, these subsystems are also modeled as DEVS models, which consist of control model and sensor/actuator Activities. The control model of each subsystem interacts with the real world through sensor/actuator Activities. These subsystem models are coupled together (by connect one model's output port to another model's input port) so they can communicate. The coupling among the models is corresponding to the connection among the subsystems in the real world.

To test the models of distributed real time systems, simulation methods are applied in our approach. For the purpose of simulation, environment model and sensor/actuator SimActivities are developed to simulate the real environment and sensor/actuator Activities. An Activity and its corresponding SimActivity share the same interfaces so the model using them can keep unchanged from simulation stage to execution stage. Different simulation methods can be applied to simulate and test the models incrementally. These simulation methods include centralized fast-mode and real-time simulation, distributed real-time simulation and hardware-in-the-loop (HIL) simulation. A more detailed description will be given in the next section. Note that each subcomponent can also be tested/simulated independently because DEVS has a well-defined concept of system modularity.

After the models are validated by simulations, they are mapped to the real hardware for execution. Similar to a stand-alone system, each subsystem needs to conduct an "Activity Mapping" to associate the sensor/actuator Activities to the corresponding sensor/actuator hardware. In addition, as the models are actually executed on different network computers, a "Model Mapping" is

needed to map the models to their corresponding host computers. These computers are physically connected by the network and they execute the models that are logically coupled together by DEVS coupling. To govern this mapping, a prototype Model Mapping Specification has been developed, which will map the models to their network nodes, while maintaining the coupling among them. As such, model continuity for distributed real time systems means not only the control model of each subsystem remain unchanged but also the coupling among the component models is maintained from the simulation to distributed execution.

In real execution, the control model of each subsystem makes decisions based on its control logic. It interacts with the real environment through sensor/actuator Activities. If a model sends out a message, based on the coupling, this message will be sent across the network and put to another model's input port. Again, with model continuity, all the subsystems will work and cooperate as were simulated.

5. SIMULATION AND TESTING OF DISTRIBUTED REAL TIME SYSTEMS

Distributed real time system is inheritly complex because the functions of the system are carried out by distributed computers over network. With our approach to model the whole system as a large coupled model, this model can be simulated and tested in our simulation framework. To enable simulation, environment model and sensor/actuator SimActivities are developed to simulate the real environment and sensor/actuator hardware. In this section, three different simulation methods are shown to give a step-wise simulation and testing of the models. These methods are central simulation, distributed simulation and hardware-in-the-loop simulation. To help to understand these methods, an example distributed real time system with two network nodes (two component models) is shown in Figure 4.

The first step is central simulation. In central simulation, the two models and environment model are all in one computer. Fast-mode simulator and real-time simulator are chosen to simulate and test the model respectively. As fast model simulation verifies system's logic behavior, real time simulation verifies system's temporal behavior.

As central simulation test models' logic and temporal behavior in one computer, it doesn't consider the network effect such as network delay. There are two ways to take account of this network factor. One way is to model the network and add the network model into central simulation. Another way is to run simulation over the real network. We adopt the second way to conduct distributed simulation of the system. In order to conduct a meaningful testing, the network the simulation is running should be the same or at least similar to the network the model will be really executed. As shown in figure 4, in distributed simulation, two models stay on two different computers. The environment model may stay on another computer or on the same computer as one of the models. The coupling between these computers remains the same, but it happens across the network. All of these models are simulated by real time simulators. These real time simulators take care of the underline network

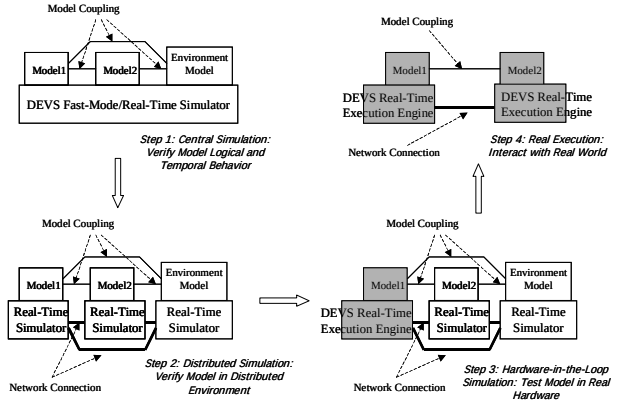

Figure 4: Simulation of Distributed Real Time System

communication so it is transparent to the model. As such, there is not need to change the model for network communication.

In distributed simulation, the real network is included so the system is simulated and tested over the real network. To further this test, real hardware the model will be executed can also be included into our simulation. This is the hardware-in-the-loop (HIL) simulation. In HIL simulation for distributed real time systems, one or more models can be distributed to their hardware to be simulated and tested. In the example of Figure 4, Model 1 along with its real-time execution engine stays on the real hardware. Model 2, environment model and their real time simulators stay on other computers. These models still keep the same coupling. However, the model on real hardware may use some or all of its sensor/actuator hardware to interact with the real world. Similar to the description in section 3, different configuration can be applied to test different aspects of the model.

After all these simulations, we have confidence that the distributed system will operate as we simulated. Then the models are mapped to the real hardware for execution. In real execution, DEVS real-time execution engine executes the model and take care of the underline

network communication. The environment model is gone, as all models interact with the real environment.

6. EMPLOYING EXPERIMENTAL FRAMES FOR TESTING

In previous sections, we have shown that simulation methods can be applied to test distributed or non-distributed real time systems in an incremental fashion. . We have discussed a testing methodology that consists of an environment model and SimActivities in which control logic can be tested. Since such testing mainly focus on the interaction between the environment model and control model, a more general testing environment can be developed using the concept of experiment frame. An experimental frame is a specification of the conditions under which a system is observed or experimented with [3]. A typical experimental frame has three types of components: *generator*, which stimulates the system under investigation in a known, desired fashion; *acceptor*, which monitors an experiment to see that desired conditions are met; and *transducer*, which observes and analyzes the system outputs. In the context of real time software design and test, an experimental frame acts as a test module to serve the functions of a test event generator, test monitor and performance analyzer. The real environment in which the application is embedded is usually modeled and included in the experimental frame. A related example can be found in [8].

With the experimental frame concept, the example shown in Figure 4 to test a distributed system can be generalized as shown in Figure 5. Here, experimental frame replaces the environment model to provide a more general and powerful testing environment. The environment model is still needed inside the experimental frame to interact with control model. However, more special *generators* can also be added into the experimental frame to provide special case test. Notice that with experimental frames, not only can the control logic be tested and validated, but also the performance of the model, such as average response time, can also be measured by using a *transducer*. Moreover attributes of intelligent behavior can be captured through specialized experimental frames and tested in the various phases of development.

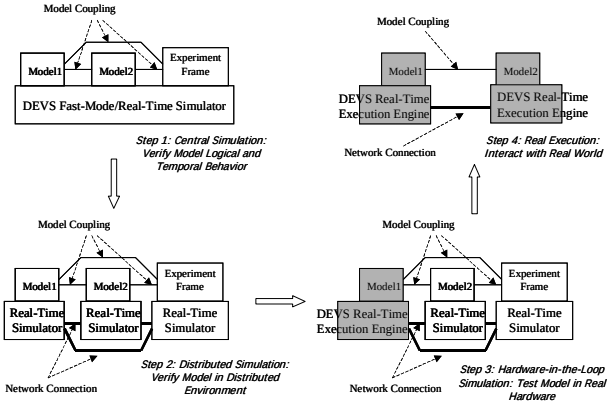

Figure 5: Testing of Distributed Real Time System Using Experimental Frame

In order to maintain model continuity, special attention has to be paid when introducing a test module to conduct testing. For example, in section 2 and 4, SimActivity has been introduced and we require that a SimActivity should have the same interfaces as an Activity. Similar restrictions are also needed when experimental frames are integrated into the system for model testing so that model continuity can be maintained.

7. CONCLUSION

Separation of models and simulators as distinct, though interacting, elements, supports model continuity. This means that the same model may be handled by different simulators appropriate to the design, testing, and execution phases of intelligent system design. Continuity of the control logic model is particularly important for design of real time, distributed intelligent systems, whose complexity would otherwise overwhelm the designers. Modeling and simulation environments, based on the DEVS formalism, can support such model continuity. An example in robotic system design has been developed and will be discussed in future papers.

8. REFERENCES

[1] Lee, E.A., "What's Ahead for Embedded Software" *IEEE Computer*, Volume: 33 Issue: 7, Sep 2000.

[2] Pimentel, A.D.; Hertzbetger, L.O.; Lieverse, P.; van der Wolf, P.; Deprettere, E.E. **"**Exploring embedded-systems architectures with Artemis" *IEEE Computer* Volume: 34 Issue: 11, Nov. 2001

[3] Zeigler, B.P., T.G. Kim, and H. Praehofer, *Theory of Modeling and Simulation. 2 ed. 2000*, New York, NY: Academic Press

[4] Slan C. Shaw, *Real-time Systems and Software*, 2001, John Wiley & Sons

[5] J.S. Hong, and T.G. Kim, "Real-time Discrete Event System Specification Formalism for Seamless Real-time Software Development," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 7, pp.355-375, 1997.

[6] Feijun Song; Folleco, A.; An, E., "High fidelity hardware-in-the-loop simulation development for an autonomous underwater vehicle" OCEANS, 2001. MTS/IEEE Conference and Exhibition, Volume: 1 , 2001

[7] Wells, R.B.; Fisher, J.; Ying Zhou; Johnson, B.K.; Kyte, M., "Hardware and software considerations for implementing hardware-in-the-loop traffic simulation " Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE , Volume: 3 , 2001

[8] Schulz, S.; Buchenrieder, K.J.; Rozenblit, J.W. **"**Multilevel testing for design verification of embedded systems" *IEEE Design & Test of Computers* Volume: 19 Issue: 2 , March-April 2002